Mumford and Shah Functional: VLSI Analysis and Implementation

Mumford and Shah functional: VLSI analysis and implementation

Maurizio Martina, *Member IEEE*, Guido Masera, *Member IEEE*

Abstract

This paper describes the analysis of the Mumford and Shah functional from the implementation point of view. Our goal is to show results in terms of complexity for real time applications, such as motion estimation based on segmentation techniques, of the Mumford and Shah functional. Moreover the sensitivity to finite precision representation is addressed, a fast VLSI architecture is described and results obtained for its complete implementation on a 0.13 μm standard cells technology are presented.

Index Terms

Image Segmentation, Mumford and Shah, Performance Evaluation, VLSI implementation

I. Introduction

Recently the use of the Mumford and Shah functional [1] has been investigated in different applications [2], [3], [4]. In [4] a novel interpretation of the optic flow has led to an extension of the Mumford and Shah functional to motion segmentation. This approach named Motion Competition is intended to join motion estimation and segmentation to derive a variational approach for the segmentation of the image domain into regions of homogeneous motion.

In terms of implementation several iterative algorithms can be found in the literature, e.g. [3], [5], [6], [7], [8], [9] and [10]. However the computational complexity behind the Mumford and Shah model has not been stressed yet. The multigrid approach is an interesting technique to speed-up iterative methods for solving elliptic problems: the basic idea is to find first a solution to the problem, solving it on a coarse mesh and then to refine the original problem on a fine

The authors are with CERCOM (Center for Multimedia Radio Communications) - Dip. di Elettronica - Politecnico di Torino, Corso Duca degli Abruzzi 24, I-10129 Torino (Italy) - corresponding author: Maurizio MARTINA - E-mail: maurizio.martina(guido.masera)@polito.it, Tel: +39 011 2276 512 / +39 011 564 4102, Fax: +39 011 564 4099

mesh. This approach has been successfully employed for the Mumford and Shah functional in [7] proving noteworthy speed-up in terms of iterations reduction. Its main drawback is the amount of memory required to store coarse solutions that will be employed during the solution of finer grids. Moreover to move from a coarser grid to a finer one and vice-versa, restriction and expansion operators are required.

Other solutions are based on the Steepest Descent (e.g. [3] and [9]) to reduce the number of iterations. Besides the Preconditioned Conjugate Gradient (PCG) method is a very effective technique for solving sparse linear equations systems as $Ax = b$. It is based on preconditioning the so called *A-orthogonality* that improves the converge to the solution employing *A-orthogonal* search directions $d_{(i)}$ ($d_{(i)}^T A d_{(j)} = 0$). Its use to speed-up the Mumford and Shah functional implementation has been addressed [10], showing interesting results. The main drawback of the PCG method is its sensitivity to finite precision representation. In fact working with high precision floating point tools (e.g `Matlab`) this phenomenon is very limited. On the contrary we are interested in fixed point implementations that could tackle PCG method effectiveness.

A simple numerical iterative solution based on non-linear Gauss-Seidel method can be employed [5] to reduce the number of iterations: the computational complexity is still proportional to the number of iterations k and to the image size $r \times c$, where r is the number of rows and c is the number of columns. Running a C model on a 3.06 GHz Intel Xeon with 2 GB of RAM it can be observed that for 255 gray levels images, 270-280 ms are required on a QCIF (176×144 pixels) frame and 1120-1130 ms on a CIF (352×288 pixels) frame. The power consumption [11], with a supply voltage of 1.5 V is estimated in 60-70W that means more than 30 μJ per sample per iteration: even with a modern processor, the computational complexity to perform real time segmentation is too inadequate for a general purpose CPU.

The contribution of this work is twofold: first to focus on the computational complexity of solving the Mumford and Shah functional, giving results on execution time and on finite precision representation effects. Second to propose a fast hardware implementation suitable for new motion estimation techniques, as Motion Competition, with high frame rate video sequences. The proposed architecture key points are: 1) a fully parallel data–path with two hardware dividers; 2) an improved diagonal scanning order to effectively feed the data–path pipeline. The complete VLSI flow performed for the proposed architecture includes the generation of actual post place and route figures in terms of complexity, area, clock frequency and power consumption.

II. THEORETICAL FRAMEWORK

The Mumford and Shah approach is based on a mathematical model that considers the segmentation problem as a partition of the image domain $\Omega \subset \mathbf{R}^2$ in open subsets Ω_i. Given an image, whose intensity function is defined as $g : \Omega \rightarrow \mathbf{R}$, the Mumford and Shah functional [1] aims to find a smooth approximation u of g in each sub-domain Ω_i:

$$E(u, K) = \int_{\Omega} (u - g)^2 dx + \alpha \int_{\Omega \setminus K} |\nabla u|^2 \, dx + \beta \parallel K \parallel \tag{1}$$

where K is the set of Ω_i boundaries and $\parallel K \parallel$ denotes K set length, α is a parameter related to the scale [9], β is a parameter related to the contrast [9], $x = (x_1, x_2)$, dx is the Lebesgue measure in the plane and $|\nabla u|^2 = \left| \frac{\partial u}{\partial x_1} + \frac{\partial u}{\partial x_2} \right|^2$. In [12] Ambrosio and Tortorelli demonstrated that the functional described in equation 1 can be approximated, in the Γ-convergence sense by

$$E_{\varepsilon}(u, z) = \int_{\Omega} (u - g)^2 + \alpha z^2 \left| \nabla u \right|^2 + \beta \Phi(\varepsilon, z) dx \tag{2}$$

with $\Phi(\varepsilon, z) = \varepsilon \left| \nabla z \right|^2 + \frac{(1-z)^2}{4\varepsilon}$, where the function z yields an approximate description of the set of curves K and $|\nabla z|^2 = \left| \frac{\partial z}{\partial x_1} + \frac{\partial z}{\partial x_2} \right|^2$. Thus the solution obtained minimizing $E_{\varepsilon}(u, z)$ tends to the solution obtained minimizing $E(u, K)$ as $\varepsilon \rightarrow 0$. As equation 2 is an elliptic problem its minimizers satisfy the Euler-Lagrange differential equation both for u and z. So that we need to: **1)** develop u and z Euler-Lagrange equations; **2)** discretize g, u and z on a uniform grid of $N \times N$ nodes with mesh size h - thus they become arrays $g_{i,j}, u_{i,j}, z_{i,j}$. Together they correspond to a nonlinear system of equations for the $2N^2$ unknown values $u_{i,j}, z_{i,j}$. Employing an iterative algorithm as the nonlinear Gauss-Seidel method [5] we obtain

$$u_{i,j} = \frac{\tilde{u}_{i,j}/h^2 + g_{i,j}/\alpha}{1/\alpha + \tilde{z}_{i,j}/h^2}, \; z_{i,j} = \frac{4\hat{z}_{i,j} + h^2/\varepsilon^2}{16 + \hat{u}_{i,j} + h^2/\varepsilon^2} \tag{3}$$

where $\tilde{u}_{i,j} = z^2_{i+1,j} u_{i+1,j} + z^2_{i-1,j} u_{i-1,j} + z^2_{i,j+1} u_{i,j+1} + z^2_{i,j-1} u_{i,j-1}$, $\tilde{z}_{i,j} = z^2_{i+1,j} + z^2_{i-1,j} + z^2_{i,j+1} + z^2_{i,j-1}$, $\hat{z}_{i,j} = z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1}$ and $\hat{u}_{i,j} = 4\alpha \left| \nabla u \right|^2_{i,j} / \beta\varepsilon$. From equation 3 it can be observed that noteworthy computational complexity is required to compute $u_{i,j}$ and $z_{i,j}$. For a complete formal derivation of equation 3 refer to [5].

III. PARAMETERS, ITERATIONS AND FINITE PRECISION EFFECTS

A. Parameters range choice

Obtaining optimal values for α, β and ε is not straightforward, however different approaches can be found in the literature [3], [9], [13] and this selection task is not a goal of this work.

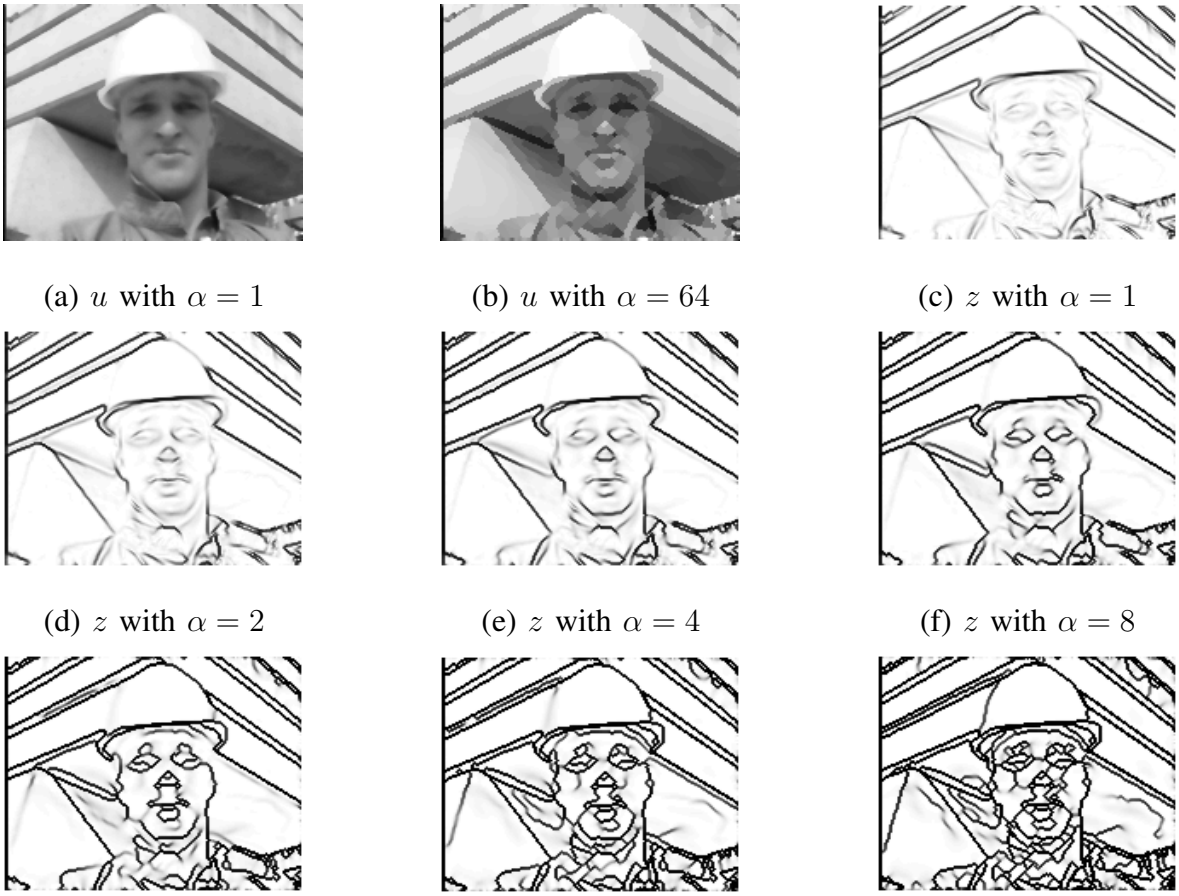
Nevertheless selecting α, β and ε as powers of 2 would lead to strongly reduce the computational complexity [14]. In this paper we will discuss the ranges suggested in [14], namely $\varepsilon \in [1/32, 1/2]$, $\beta \in [1/256, 1/2]$ and $\alpha \in [1, 64]$ with the image values in the range $[0, 1]$. Figure 1 shows the results obtained varying α as a power of 2 in $[1, 64]$ with $h = 1$, $\varepsilon = 1/16$ and $\beta = 1/256$ on the image "foreman": a wide spectrum of images from smoother to sharper are covered. Figure 2 shows the mean error obtained varying ε and β as powers of 2 in the ranges $[1/32, 1/2]$ and $[1/256, 1/2]$ respectively, with $\alpha = 8$ for the same image: it can be noticed that the error is rather limited; moreover a similar behavior has been observed for several other images and parameters choices, thus the performed analysis shows that the choice of α, β and ε as powers of 2 results in acceptable approximations of the Mumford and Shah functional.

The solution of equations 2 with the non linear Gauss-Seidel method implies certain initial conditions (i.e. the values at iteration 0), that can be chosen, according to [5], as $u_{i,j}^{(0)} = g_{i,j}$ and $z_{i,j}^{(0)} = 1$. Moreover a proper number of iterations is required to obtain good results. From the direct implementation of equation 3 on some QCIF and CIF images, with $h = 1$, $\alpha = 8$, $\beta = 1/256$ and $\varepsilon = 1/16$, we can observe that: after approximately 20 iterations the increase of accuracy on u and z (from the k^{th} to the $k + 1^{th}$ iteration) tends to saturate in terms of Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR). In figure 3 this behavior is shown for different images ("Foreman", "Mobile", "Tempete" and "Paris") together with the mean value (dashed curves). The solid curves, representing the PSNR obtained from the mean MSE, better show the saturation phenomenon.

In the following we will fix the Mumford and Shah functional parameters as $h = 1$, $\alpha = 8$, $\beta = 1/256$, $\varepsilon = 1/16$ and the maximum number of iterations to 20: with this set of values we will concentrate on the Mumford and Shah functional (equation 3) sensitivity to finite precision effects.

B. Finite precision effects

Since $g_{i,j}$, $u_{i,j}$ and $z_{i,j}$ are in the range $[0, 1]$, and $\alpha \geq 1$, the following inequalities hold true for the numerator (Nu) and the denominator (Du) of $u_{i,j}$ in equation 3: $0 \leq Nu \leq 4 + 1/\alpha$ and $1/\alpha \leq Du \leq 4 + 1/\alpha$, so that, 3 bits are needed for the integer part of Nu and Du. Analogously we have: $1/\varepsilon^2 \leq Nz \leq 16 + 1/\varepsilon^2$ and $16 + 1/\varepsilon^2 \leq Dz \leq 16 + 2\alpha/\beta\varepsilon$, where Nz and Dz are the numerator and the denominator of $z_{i,j}$ in equation 3. Since $\alpha \geq 1$, $0 \leq \beta \leq 1$ and $0 \leq \varepsilon \leq 1$

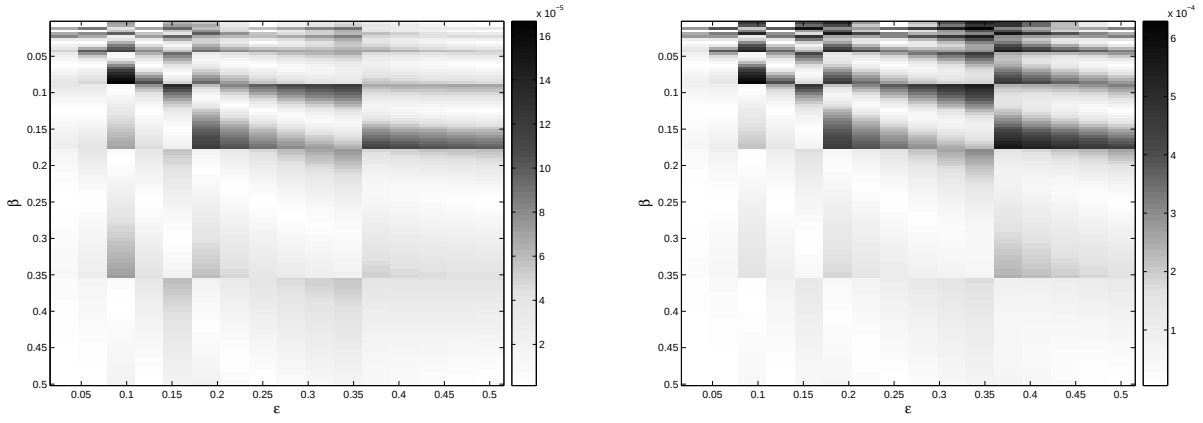


(a) u with $\alpha = 1$ (b) u with $\alpha = 64$ (c) z with $\alpha = 1$

(d) z with $\alpha = 2$ (e) z with $\alpha = 4$ (f) z with $\alpha = 8$

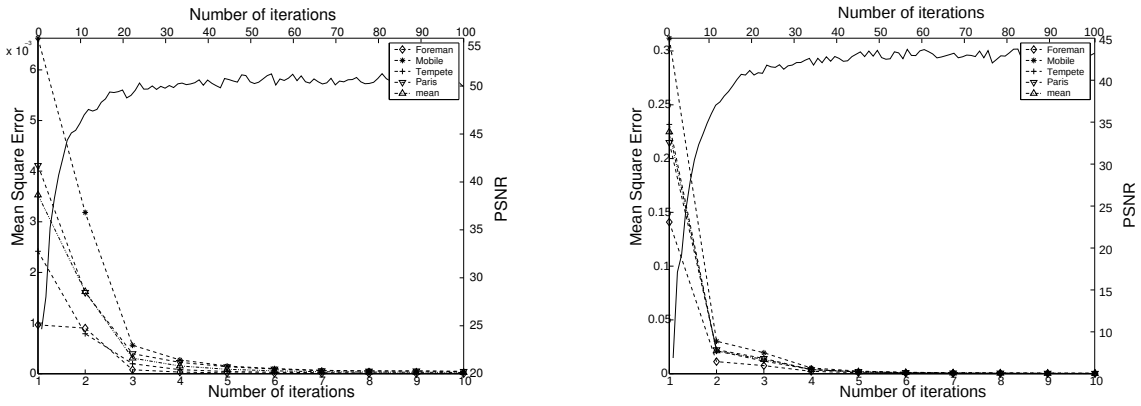
(g) z with $\alpha = 16$ (h) u with $\alpha = 32$ (j) z with $\alpha = 64$

Fig. 1. Mumford and Shah results on the QCIF image "Foreman" with $\alpha \in [1, 64]$



(a) β and ε error analysis on u (b) β and ε error analysis on z

Fig. 2. Error analysis on β and ε



(a) u MSE and mean PSNR Vs number of iterations (b) z MSE and mean PSNR Vs number of iterations

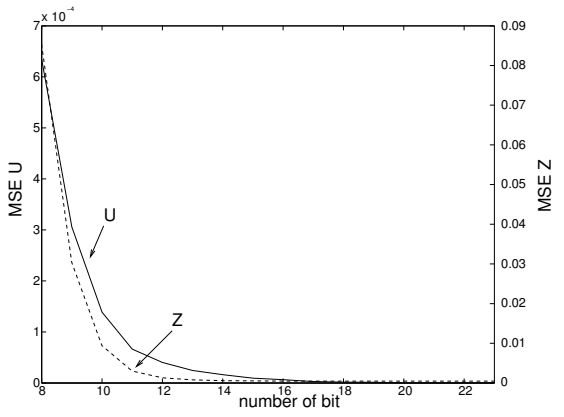
Fig. 3. u and z accuracy increase and the mean PSNR Vs number of iterations for the QCIF image "Foreman", the CIF images "Mobile", "Tempete","Paris"

the number of bits for correctly representing the integer part of Nz and Dz would be rather higher than the number obtained for the integer part of Nu and Du. In particular with the worst contributions of α, β and ε, around 20 bits for the integer part of Dz are needed. To reduce this number we rewrite equation 3 ($z_{i,j}$) with $h = 1$ as

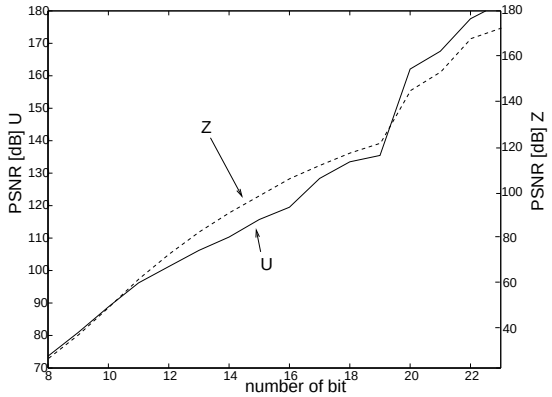
$$z_{i,j} = \frac{\frac{1}{\alpha} + 4\frac{\varepsilon^2}{\alpha}\hat{z}_{i,j}}{\frac{1}{\alpha} + 16\frac{\varepsilon^2}{\alpha} + 4\frac{\varepsilon}{\beta}\left|\nabla u\right|^2_{i,j}} \tag{4}$$

Now we have: $1/\alpha \leq Nz \leq 16\varepsilon^2/\alpha + 1/\alpha$ and $1/\alpha + 16\varepsilon^2/\alpha \leq Dz \leq 1/\alpha + 16\varepsilon^2/\alpha + 2\varepsilon/\beta$. With the range considered for α, β and ε, Nz and Dz need at least 3 and 9 bits respectively for the integer part. Although different choices of parameters could also be of interest, in the rest of the paper analysis and synthesis results will be limited to the case $h = 1$, $\alpha = 8$, $\beta = 1/256$ and $\varepsilon = 1/16$: the `Matlab` implementation of the Mumford and Shah functional in this case will be assumed as the reference model and used as a comparison term while investigating the finite precision effects of $u_{i,j}$ and $z_{i,j}$ calculation.

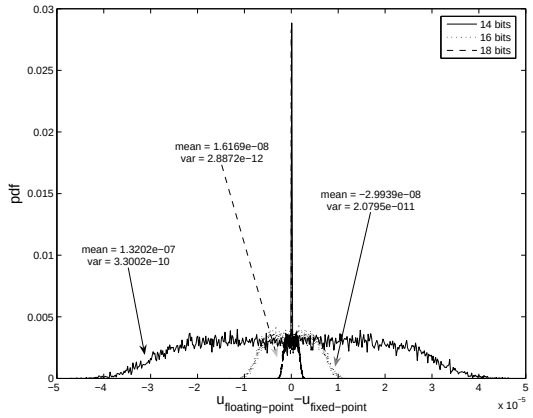
In figure 4 (a, b) mean values for MSE and PSNR versus the number of bits devoted to the fractional part representation (m) over different images are shown. After 16 bits for the fractional part the difference between the floating point *reference model* and the finite precision implementation is negligible. Moreover after $m = 18 - 20$ bits, the values for the PSNR might not be completely reliable due to the joint effect of finite precision and algorithm iterations.



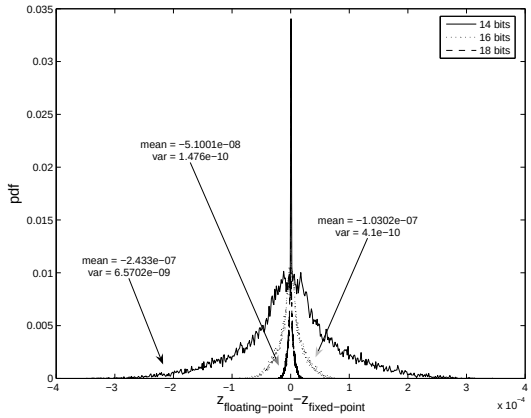
(a) u and z MSE Vs m



(b) u and z PSNR Vs m



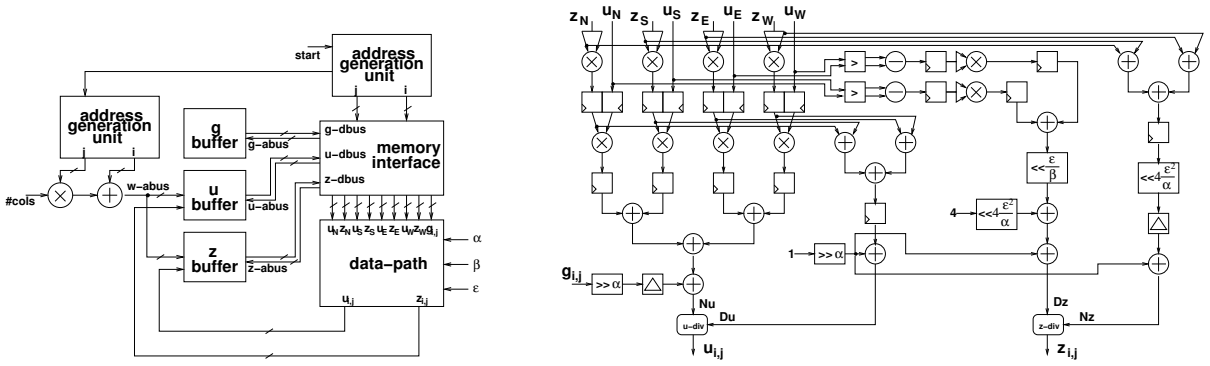
(c) u error distribution with $m = 14, 16, 18$



(d) z error distribution with $m = 14, 16, 18$

Fig. 4. mean u and z difference between the floating point *reference model* and finite precision C model Vs number of bits for the fractional part (m) (a), (b). Error distribution on u and z with $m = 14, 16, 18$ (c), (d) for the QCIF image "foreman"

Furthermore in figure 4 (c, d) the error distributions on u and z for $m = 14, 16, 18$ are shown. These results have been obtained calculating the difference between the floating point results and the fixed point ones, and represent the error distribution (pdf). The frame used is "foreman" and the parameters values are $h = 1$, $\alpha = 8$, $\beta = 1/256$ and $\varepsilon = 1/16$. It is worth noticing that similar results have been obtained for different choices of parameter values in the aforementioned ranges. From the results shown in figure 4 we can conclude that a hardware dedicated implementation, employing $m \geq 16$ would be able to achieve very good performance.



(a) Proposed architecture block scheme

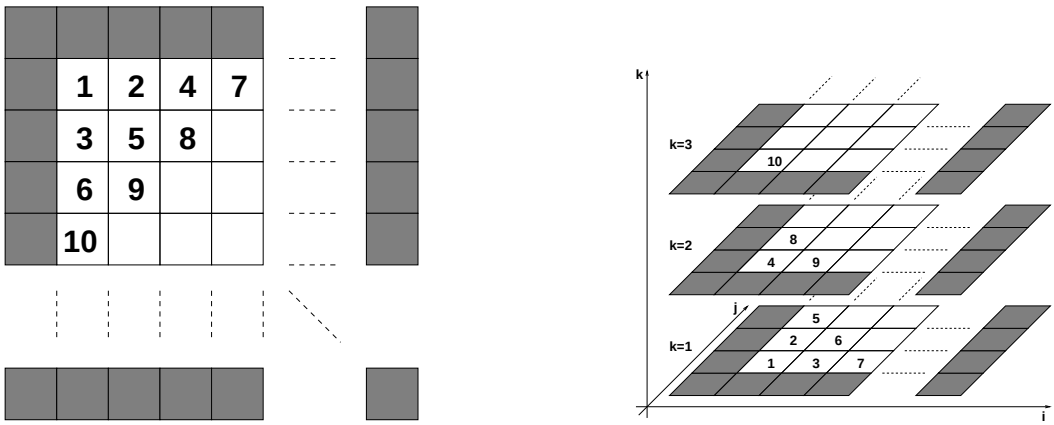
(b) Data–path block scheme

Fig. 5. Proposed architecture block scheme (a) and detail of the the data–path (b)

IV. Implementation issues

In [14] software implementations on modern microprocessors of the Mumford and Shah functional are addressed, showing that high frame rate sequences cannot be processed in real–time; in fact to perform 20 iterations on a QCIF frame 4 s and 12.5 s on the high performance Texas Instruments TMS320C6711 and on the StrongARM SA-1100 respectively. Thus in a system where real time processing is needed, as the video scenario, a dedicated VLSI implementation would be required. From the analysis performed in section III it can be observed that it is difficult to fix the values of the Mumford and Shah functional parameters, the number of iterations and the number of bits m devoted to the representation of the fractional part of the data. To obtain a highly reusable architecture a parametric VHDL model has been developed, focusing on high performance in terms of number of sustained frame, while granting high quality of the resulting u and z.

In the literature the implementation of the Gauss-Seidel method for the solution of equations systems has already been addressed. However solutions proposed in the literature can not be straightforwardly adapted to the case of the Mumford and Shah functional. In fact many works describe the implementation of the Gauss-Seidel method with particular emphasis on serial and shared memory parallel computing [15], [16] or on systolic solutions [17]. As pointed out in [18] there is a strong data dependency between the pixels produced with the Gauss-Seidel algorithm. In the following we will refer to the neighbors of the considered pixel as *north, south, east* and



(a) Diagonal scanning order

(b) Improved diagonal scanning order

Fig. 6. Pixel scanning order

west: $x_{i-1,j} = x_N$, $x_{i+1,j} = x_S$, $x_{i,j+1} = x_E$ and $x_{i,j-1} = x_W$ where x can be either u or z.

The proposed architecture, depicted in figure 5 (a), is composed by: **a)** three buffers to store respectively the original image g, the approximation u and the edges z; **b)** two address generation units devoted to manage the correct scanning order to elaborate the samples; **c)** a complex data–path to implement equations 3 and 4; **d)** a simple memory interface unit to load all the data required to perform the operations described by equations 3 and 4.

A. Address unit

As it can be observed from equations 3 and 4 $u_{i,j}$ (and $z_{i,j}$) depends on north, south, east and west values. Thus scanning the image by rows and columns (or vice-versa by columns and rows) to evaluate the new value of $u_{i,j+1}$ the computation of $u_{i,j}$ and $z_{i,j}$ must be completed. In fact for $u_{i,j+1}$ the pixels $u_{i,j}$ and $z_{i,j}$ act as u_W and z_W respectively. Instead of scanning the image row-wise and column-wise (or vice-versa) a "diagonal" scanning order can be employed (figure 6 (a)): a high latency for generating the first values is required, while for the following pixels the latency decreases as the pipeline is filled. Moreover the algorithm iterative nature can be exploited to pipeline also on the iterations [18]. This improved diagonal scanning order is shown in figure 6 (b) where i and j represent rows and columns directions and k represents the iteration step; the number associated to each pixel represents the order used to process the

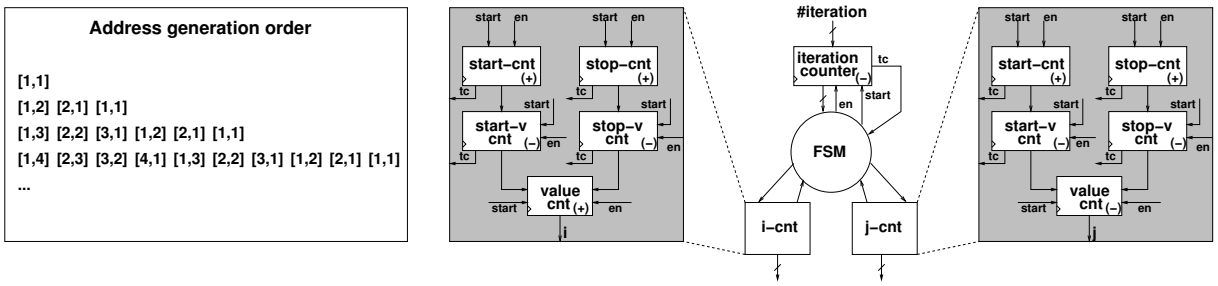


Fig. 7. Address generation unit

different values. This scanning order leads to the generation of the addresses depicted in figure 7 on the left; they can be generated resorting to some counters and small control logic, as depicted in figure 7 on the right, where $start - cnt$ and $stop - cnt$ are up-counters to generate rows (i) and columns (j) current index limits. $start - vcnt$ and $stop - vcnt$ are down-counters to generate new index limits taking into account both the diagonal order and the algorithm iterations. $value - cnt$ are an up-counter (i) for row indices generation and a down-counter (j) for columns indices generation. The finite state machine reads counters terminal count (tc) signals and generates $start$ and $enable$ (en) signals also checking rows and columns bounds. Finally $iteration\ counter$ is a down-counter devoted to perform the number of iterations programmed. This architecture has been synthesized with Synopsys `design compiler`, placed and routed with Cadence `encounter` on a 0.13 μm standard cell technology both for QCIF and CIF frames. The address generation unit for QCIF frames can run at 550 MHz requiring an area of 9170 μm^2, whereas for CIF frames can run at 510 MHz requiring an area of 10655 μm^2.

B. Data–path

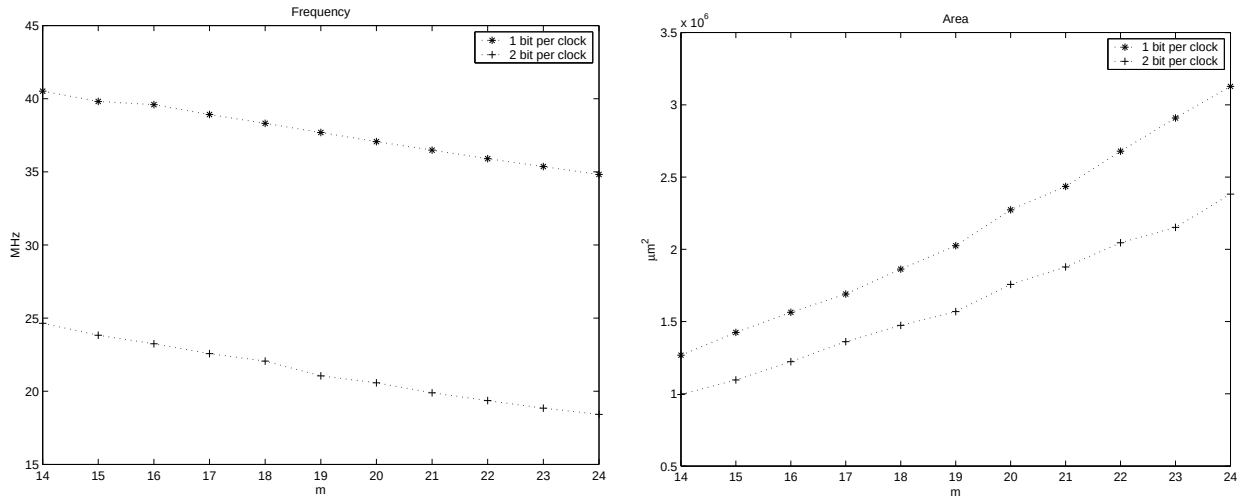
The proposed architecture implements equations 3 and 4 respectively to generate both $u_{i,j}$ and $z_{i,j}$. As it can be inferred from equations 3 and 4, the architecture bottleneck is the division. In the case of the proposed architecture the division algorithm needs to be unrolled and pipelined in order to obtain a parallel divider able to sustain high throughput. Moreover, since $u_{i,j}$ and $z_{i,j}$ are in $[0, 1]$ the numerator will never be greater than the denominator, thus we do not need to introduce operands normalization, as in most SRT dividers implementations [19].

As depicted in figure 5 (b), to perform these operations, the data–path can process at the same time u_N, u_S, u_W, u_E, z_N, z_S, z_W, z_E and $g_{i,j}$. As described by equations 3 and 4 z_N, z_S, z_E

and z_W can be employed to evaluate the square values required by $\tilde{u}_{i,j}$ and $u_{i,j}$ denominator; moreover they are needed to calculate $\hat{z}_{i,j}$. Employing four multipliers the four square values can be calculated at the same time (top-left in figure 5 (b)), meanwhile the four z values are added together to obtain $\hat{z}_{i,j}$ (top-right in figure 5 (b)). During the same clock cycle u_N, u_S, u_E and u_W are employed to evaluate the first part of the discrete gradient $|\nabla u|^2_{i,j}$ and are stored into four registers. Since the discrete gradient is squared and multiplied by four it can be computed simply adding together the squared differences, in fact the following expression holds true $4\,|\nabla u|^2_{i,j} = (u_S - u_N)^2 + (u_E - u_W)^2$. The discrete gradient calculation has been split into three parts: **1)** subtractions $u_{SN} = u_S - u_N$ and $u_{EW} = u_E - u_W$, **2)** squares $u_y = u^2_{SN}$ and $u_x = u^2_{EW}$, **3)** addition $u_y + u_x$, where each part is implemented in a separate clock cycle (top-center in figure 5 (b)). While the discrete gradient second part is evaluated, the squared z values are added together to obtain $u_{i,j}$ denominator and multiplied by the proper u input (u_N, u_S, u_E or u_W) to compute the partial values required by $\tilde{u}_{i,j}$. Moreover in the same clock cycle $\hat{z}_{i,j}$ is shifted by $4\varepsilon^2/\alpha$ (see figure 5 (b) in the middle).

During the following clock cycle the evaluation of $\tilde{u}_{i,j}$ is completed and $g_{i,j}$ (shifted by α) is added to generate $u_{i,j}$ numerator. In the same clock cycle the computation of $u_{i,j}$ denominator is completed adding to z squared values the term $1/\alpha$. Moreover $z_{i,j}$ numerator and denominator are calculated: the former adding to $4\varepsilon^2/\alpha\hat{z}$ the term $1/\alpha$, the latter shifting the discrete gradient by ε/β and adding $1/\alpha + 16\varepsilon^2/\beta$. Finally two dividers are employed to compute in parallel $u_{i,j}$ and $z_{i,j}$ starting from their numerators and denominators (bottom part of figure 5 (b)).

From the results shown in figure 4 it is worth noticing that probably more than 12 bits will be employed for the representation of data fractional part. Since $u_{i,j}$ and $z_{i,j}$ are in $[0, 1]$, the data–path has been implemented (synthesized, placed and routed) varying the data width representation (m) from 14 to 24 bits. Besides considering the ranges detailed in section III for α, β and ε, internal data need up to 9 bits for the integer part representation. Results detailed in figure 8 show how the frequency achievable by the data–path changes varying the number of bits to represent the output results. From the results reported in figure 8 it can be observed that to make the data–path able to process a new sample at each clock cycle, the address generation unit needs to be connected to a fast memory interface block.

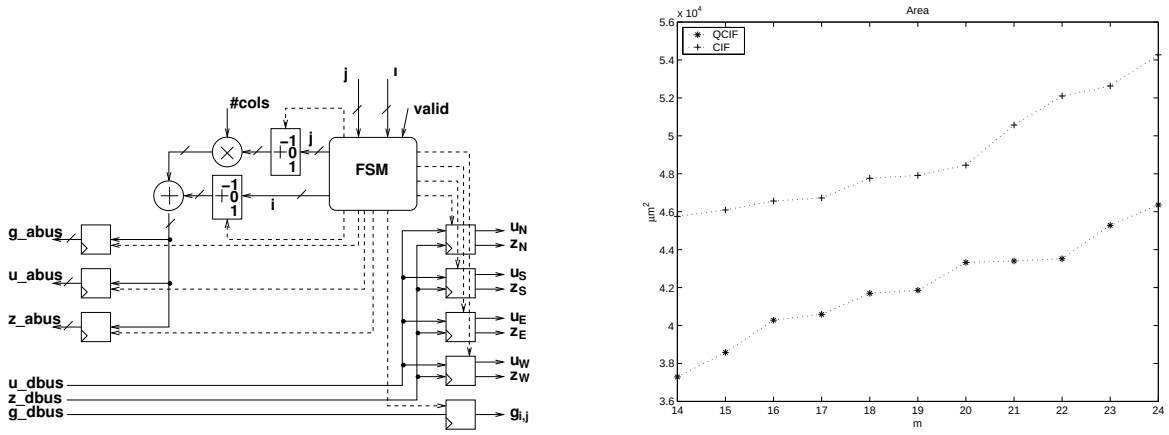


(a) Frequency (b) Area

Fig. 8. Proposed data–path: area Vs number of bit per output results (m)

C. Memory interface block

It is reasonable to suppose that u, z and g values are stored (at least partially) in three buffers. The memory interface block is a simple finite state machine that starting from the matrix row address (i) and the matrix column address (j) generated by the address generation unit, is able to load u_N, u_S, u_E, u_W, z_N, z_S, z_E, z_W and $g_{i,j}$ from the buffers and to feed the data–path with these values. Exploiting the lower frequency achievable by the data–path, the memory interface block behaves as a sort of serial to parallel component. In figure 9 (a) a block scheme of the memory interface block is shown. The finite state machine (FSM) is devoted to receive the row and column indices, i and j, calculated by the address generation unit. Starting from these values the FSM selects to add 1, 0 or -1 to in order to implement $i+1$, i or $i-1$ (analogously with j). The result obtained on the column index j ought to be multiplied by the number of columns (#COLS) and added with the result obtained on the row index i; when an address is correctly generated it is validated by the FSM. Given an address on the proper address bus (g_abus, u_abus and z_abus), the data buffers (g, u and z) latch the data required by the memory interface block on the data bus (g_dbus, u_dbus and z_dbus); finally the FSM loads each value into its register (bottom-right in figure 9 (a)). The memory interface block has been synthesized, placed and routed on a 0.13 μm standard cell technology for QCIF and CIF frames varying m



(a) Memory interface block scheme

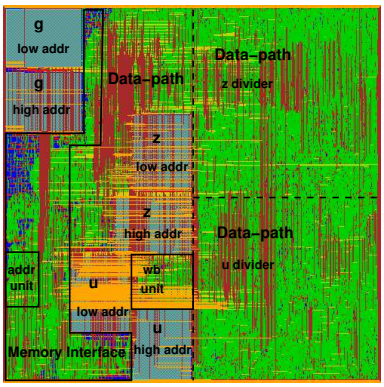
(b) Memory interface block area Vs data width

Fig. 9. Memory interface block scheme and area Vs data width for QCIF and CIF frames

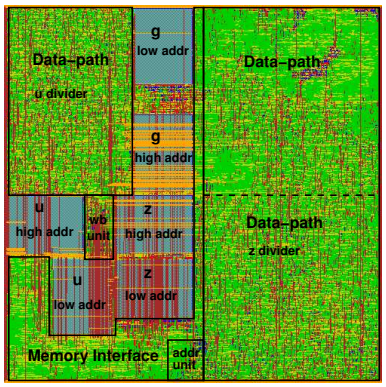
Since the critical path (for this implementation) is in the address generation ($j \times \#COLS + i$), the maximum achievable clock frequency depends only on the frame size and not on the data width. In fact the maximum achievable frequency is about 215 MHz and 190 MHz for QCIF and CIF frames respectively. Experimental results show that the amount of gates required increases as the data width grows (see figure 9 (b)). Finally, since the data–path produces two data per clock cycle, namely $u_{i,j}$ and $z_{i,j}$, they can be sent to the proper buffer resorting to a further address generator a multiplier and an adder.

V. Proposed architecture performance

In the following we will focus on the case of $m = 16$ and one bit per clock cycle, and we will discuss the global architecture performance. Simulations performed on the proposed architecture VHDL description show that the data–path needs 20 clock cycles to evaluate a new couple of results (u and z). Due to the improved diagonal scanning the data–path latency decreases as $\sum_{i=1}^{Lat_0} i$. In fact after the 20th column is processed, the simulation shows that the architecture produces a couple of new values ($u_{i,j}$, $z_{i,j}$) at each clock cycle. Considering post place and route frequency and that the memory interface should run faster than the data–path to fetch the required data, the proposed architecture can sustain more than 25 QCIF frames per second and about 15 CIF frames per second. In figure 10 (c) the performance achievable with the proposed



(a) QCIF



(b) CIF

	QCIF (144 x 176)	CIF (288 x 352)
Proposed architecture 1 iteration	676 µs	3 ms
Proposed architecture 20 iterations	13.5 ms	61 ms
[15] 20 iterations	~530 ms	~2.1 s
[7] λ in [10, 1000]	~393 ms	~1.74 s

(c) Performance comparison

Fig. 10. Post place and route architectures for QCIF and CIF frames (a) and (b). Performance comparison with [14] and [6]

architecture are summarized and compared with the run time of the fixed point C model [20] described in [14] and of a competitive software implementation [6] within the Megawave package (Megawave 2.31a) [21] running on a Linux Mandrake 10 - Pentium IV at 1.6 GHz with 512 MB of RAM. Post place and route of the whole architecture shows that 4.72 mm^2 and 4.73 mm^2 are required for QCIF and CIF frames respectively (see figure 10 (a, b)).

Thanks to the parametric VHDL description the two implementations have been achieved simply changing some parameters in the VHDL source and re-performing the design flow. In figure 10 (a, b) it can be observed that: since the QCIF design is slightly smaller than the CIF one, also the QCIF chip is slightly less dense than the CIF one. As far as the power consumption is concerned post place and route analysis shows that about 337 mW are required for the QCIF architecture and 395 mW for the CIF architecture.

VI. Conclusions

In this paper the Mumford and Shah functional has been investigated from the implementation point of view. A detailed study of finite precision representation effect has been carried out, a fast VLSI architecture has been described and results obtained through the implementation on a 0.13 μm standard cells technology have been presented. Finally the authors would like to thank the reviewers for their suggestions that have actually improved the quality of this work.

REFERENCES

[1] D. Mumford and J. Shah, "Optimal approximations by piecewise smooth functions and associated variational problems," *Comm. Pure Appl. Math.*, vol. 42, pp. 577–685, 1989.

[2] F. Sroubek and J. Flusser, "Multichannel blind iterative image restoration," *IEEE Trans. on Image Processing*, vol. 12, no. 9, pp. 1094–1106, sep. 2003.

[3] A. Brook, R. Kimmel, and N. A. Sochen, "Variational restoration and edge detection for color images," *Journal of Mathematical Imaging and Vision*, vol. 18, no. 3, pp. 247–268, 2003.

[4] D. Cremers and S. Soatto, "Motion competition: A variational approach to piecewise parametric motion segmentation," *International Journal of Computer Vision*, vol. 62, no. 3, pp. 249–265, 2005.

[5] R. March, "Visual reconstruction with discontinuities using varaitional methods," *Image and Vision Computing*, vol. 10, pp. 30–38, 1992.

[6] G. Koepfler, C. Lopez, and J. M. Morel, "A multiscale algorithm for image segmentation by variational method," *SIAM J. Numer. Anal.*, vol. 31, no. 1, pp. 282–299, feb. 1994.

[7] D. Cremers, F. Tishhäuser, J. Weickert, and C. Shnörr, "Diffusion snakes: introducing statistical shape knowledge into the Mumford and Shah functional," *International Journal of Computer Vision*, vol. 50, no. 3, pp. 295–313, 2002.

[8] T. P. Vogl, J. W. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, "Accelerating the convergence of the back-propagation method," *Biological Cybrnetics*, vol. 59, pp. 257–263, 1988.

[9] W. Vanzella, F. A. Pellegrino, and V. Torre, "Self-adaptive regularization," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 804–809, jun. 2004.

[10] F. Gibou, D. Levy, C. Cardenas, P. Liu, and A. Boyer, "Partial differential equation based segmentation for radiotherapy treatment planning," *Mathematical Biosciences and Engineering*, vol. 2, pp. 209–226, 2005.

[11] Intel, "http://www.intel.com/design/xeon/datashts/252135.htm."

[12] L. Ambrosio and V. M. Tortorelli, "Approximation of functionals depending on jumps by elliptic functionals via Γ-convergence," *Comm. Pure Appl. Math.*, vol. 43, pp. 999–1036, 1990.

[13] T. F. Chan and L. A. Vese, "Active contours without edges," *IEEE Trans. on Image Processing*, vol. 10, no. 2, pp. 266–277, feb. 2001.

[14] M. Martina and G. Masera, "Mumford and Shah functional: Finite precision analysis and software implementation," in *IEEE International Symposium on Signal Processing and Information Technology*, 2004.

[15] M. F. Adams, "A distributed memory unstructured Gauss-Seidel algorithm for multigrid smoothers," in *ACM/IEEE Proceedings of SC2001: High Performance Networking and Computing*, 2001, pp. 1–4.

[16] M. M. Strout, L. Carter, J. Ferrante, J. Freeman, and B. Kreaseck, "Combinig performance aspects of irregular Gauss-Seidel via sparse tiling," in *15th Workshop on Languages and Compilers for Parallel Computing*, 2002, pp. 1–4.

[17] F. Z. Hadjam, A. Rahmoun, and M. Benmohammed, "On designing a systolic network for the resolution of linear systems using the Gauss-Seidel method," in *IEEE International Conference on Computer Systems and Applications*, 2001, pp. 283–286.

[18] J. A. Yang and Y. Choo, "Formal derivation of an efficient parallel 2-D Gauss-Siedel method," in *Proceedings of the 6th International Parallel Processing Symposium*, 1992, pp. 204–207.

[19] J. E. Robertson, "A new class of digital division methods," *Trans. on Electronic Computers*, vol. 7, pp. 218–222, 1958.

[20] M. Martina, "Mumford and Shah C model," downloadable at www.vlsilab.polito.it/∼martina.

[21] MegaWave2, "available at: http://www.cmla.ens-cachan.fr/cmla/megawave/index.html."